Assignment 2: The Deutsch-Jozsa algorithm

Lingyu Gong

I. Lab purpose

The same as the previous experiment, going to familiarize with quantum algorithms and so on, is characterized by the advantages of quantum algorithms that can be experienced in this algorithm.

II. The Deutsch-Jozsa algorithm

1. Definition (My Understanding.)

The Deutsch-Jozsa algorithm is valuable in that it is an algorithm that can be tested to show the difference between quantum and classical algorithms, and it can be queried for multiple problems simultaneously.

Ignoring the large number of formula explanations, it can be said that the algorithm is faster and more accurate than the more classical algorithms.

2. Problem Description

Deutsch-Jozsa problem
Input: a function f: $\{0,1\}^n \rightarrow \{0,1\}$
Promise: f is either constant or balanced
Output: 0 if f is constant, 1 if f is balanced

III. Lab Implement

First, I found the code given in Qiskit and read it to understand and reproduce the functionality. The structure of the code is summarized below:

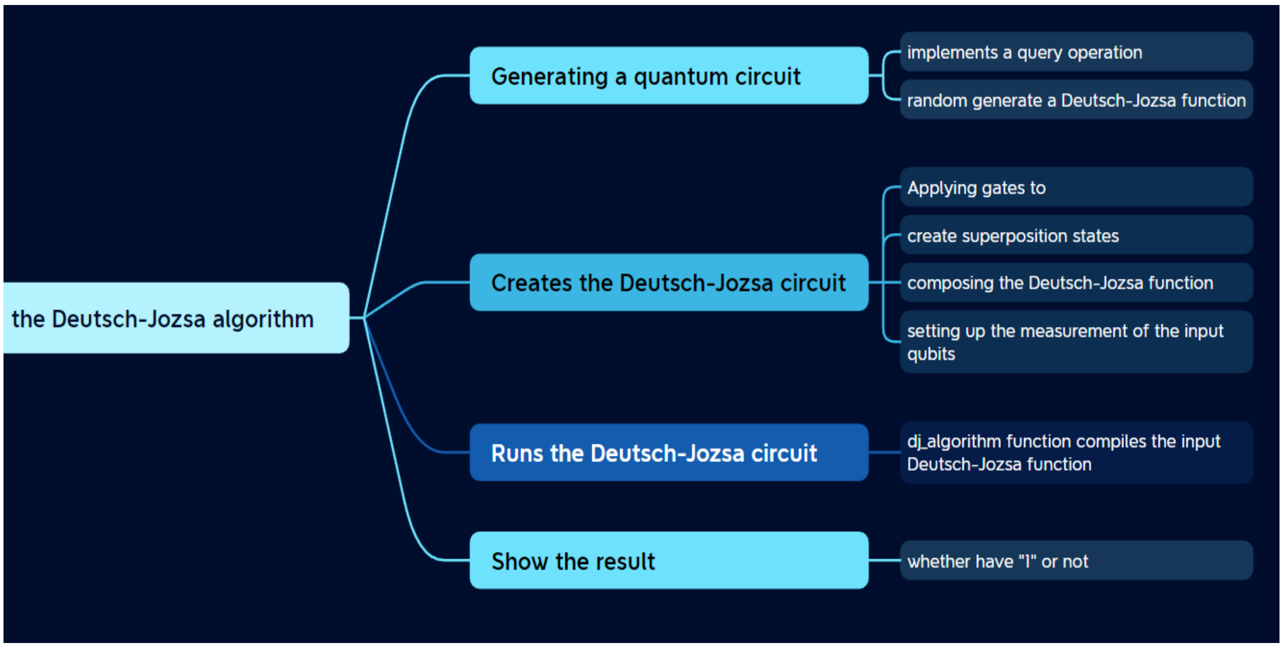

the Deutsch-Jozsa algorithm
Generating a quantum circuit
implements a query operation
random generate a Deutsch-Jozsa function
Creates the Deutsch-Jozsa circuit
Applying gates to
create superposition states
composing the Deutsch-Jozsa function
setting up the measurement of the input qubits
Runs the Deutsch-Jozsa circuit
dj_algorithm function compiles the input Deutsch-Jozsa function
Show the result
whether have "1" or not

Figure 1 The Structure of implement the algorithm.

To be more specific, the whole program is as below. The comment I used to explain the code without write to be briefly.

Step 1:

```
# Import necessary libraries:
from qiskit import QuantumCircuit
import numpy as np

# Define the 'dj_function'
def dj_function(num_qubits):# num_qubits: the number of qubits in the quantum circuit

# Create a quantum circuit
  qc = QuantumCircuit(num_qubits + 1) #num_qubits input qubits and an output qubit.

  if np.random.randint(0, 2): # Randomly flip the output qubit with a 50% chance
    qc.x(num_qubits)

  if np.random.randint(0, 2): #Randomly return a constant circuit with a 50% chance

    return qc

# Choose half of the possible input states
  on_states = np.random.choice(

    range(2**num_qubits),  # numbers to sample from

    2**num_qubits // 2,  # number of samples

    replace=False,  # makes sure states are only sampled once

  ) # promise function will behave differently.

# Define a helper function add_cx
  def add_cx(qc, bit_string): # qc and a bit string as input

    for qubit, bit in enumerate(reversed(bit_string)):

      if bit == "1":

        qc.x(qubit)

    return qc

# For each selected state, perform the following operations:
  for state in on_states:

    qc.barrier()

    qc = add_cx(qc, f"{state:0b}")

    qc.mct(list(range(num_qubits)), num_qubits)

    qc = add_cx(qc, f"{state:0b}")

# Separate the created function from the output qubit
  qc.barrier()

  return qc
```

Step 2:

```
def compile_circuit(function: QuantumCircuit):
    n = function.num_qubits – 1 # Calculate the number of qubits in the input function
    qc = QuantumCircuit(n + 1, n)  # Create a new quantum circuit for compilation, the
#last qubit is reserved for the output, and n classical bits to store measurement results.
    qc.x(n) # ensure it starts in the state |1>.
    qc.h(range(n + 1))
    qc.compose(function, inplace=True)#Compose the input function with the compiled
#circuit
    qc.h(range(n))
    qc.measure(range(n), range(n)) #Measure the input qubits and return the compiled
#circuit
    return qc
```

Step 3:

```
from qiskit_aer import AerSimulator
def dj_algorithm(function: QuantumCircuit)
    qc = compile_circuit(function)
# shots=1: this indicates that the quantum circuit is run once.
# memory=True: store the measurement outcomes in memory for later analysis.
    result = AerSimulator().run(qc, shots=1, memory=True).result()
    measurements = result.get_memory() # check 1 or 0
    if "1" in measurements[0]:
        return "balanced"
    return "constant"
```

IV. Outcome explanation

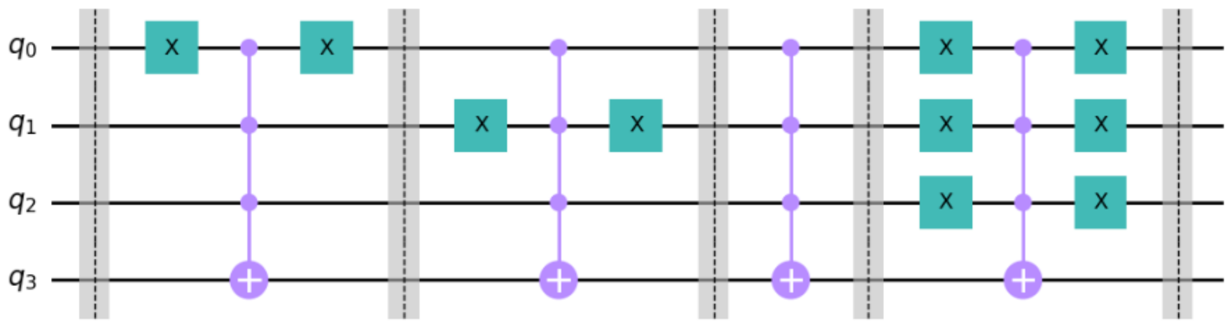
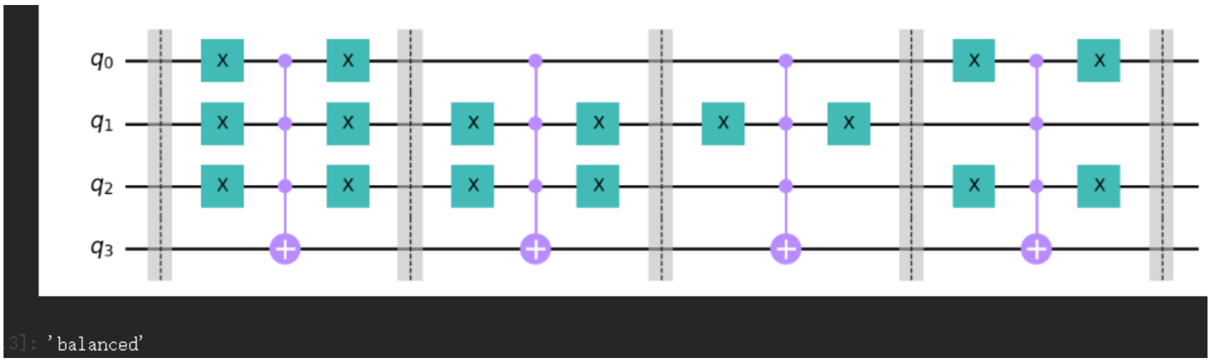


Figure 2 The whole circuit diagram

So, I change this place as “shot = 2” to make it run twice:

```
result = AerSimulator().run(qc, shots=2, memory=True).result()
```

The result is as below.



```
3]: 'balanced'
```

Figure 3 The result diagram as “shot = 2”

V. Summary

The Deutsch-Jozsa algorithm is a significant quantum algorithm that showcases the power of quantum parallelism. It can solve certain problems much faster than classical algorithms. Even though the Deutsch problem seems easy, this algorithm built the groundwork for more complex quantum algorithms and introduced key quantum computing concepts like quantum oracles and superposition.

All in all, the Deutsch-Jozsa algorithm is a brief yet illustrative example of how quantum computing can surpass classical computing in certain tasks.

Summarize the step to achieve this algorithm:

1. Prepare the input qubits in a superposition state by applying Hadamard gates.
2. Query the quantum oracle with the superimposed input states. The oracle encodes information about the function's behaviour into the quantum state.
3. Apply additional Hadamard gates to the input qubits to amplify the effects of phase kickback.
4. Measure the input qubits.
5. Analyse the measurement outcomes. If all measurements result in the same value (all 0s or all 1s), the function is constant. If at least one measurement result differs, the function is balanced.